This is a C++ program to find big numbers. The biggest known prime number ever found at this time is $M_{136,279,841}$, which is equal to $2^{136,279,841} - 1$.

Mersenne primes are numbers of the form 2^p-1, with a known prime number *p*. It is theorized that the number of Mersenne primes is infinite. Due to the existence of Mersenne primes, it is much easier to find very big prime numbers. There are currently 52 known terms in the Mersenne sequence, and the 7 biggest known prime numbers are Mersenne primes. It is important to note that not all prime numbers can serve as the exponent *p* to produce new Mersenne primes, making the search for bigger primes more and more challenging.

Even with the knowledge of Mersenne primes, blindly checking for new Mersenne primes with known primes would take an unreasonable number of computations. To make the search more efficient, this program makes use of two algorithms: the Fermat Primality test and the Lucas-Lehmer Primality test.

Computing the full value of $M_p = 2^p$-1 would take an extremely large number of computations (given that p is a known prime number e.g. *p* = 136,279,841 in $M_{136,279,841}$). Since the computations to check if that value is in fact a prime number will also be costly, some optimization is necessary. The Fermat Primality test acts as a filter to handle this problem.

The Fermat Primality test only requires a single modular exponentiation, which is much less expensive in computations than any test that fully verifies that a number is prime. With base a=3, the test is as below. The restriction on the base is that it must not be divisible by *p* and (1 < a < *p*-1).

$$3^{M_p} - 1 \equiv 1 \pmod{M_p}$$

The program computes a set number of M_p candidates with different *p* values (set to 10 in the code but can be changed to any number), runs them through the Fermat Primality test, and the numbers that pass are called Fermat Pseudoprimes. This is because as fast as the Fermat Primality test is compared to other primality tests, it also produces false positives. However, this still means that the numbers eliminated by the Fermat Primality test must

have been composite numbers, and only the pseudoprimes that passed move on to the much more costly test, the Lucas-Lehmer Primality test.

The Lucas-Lehmer test is a primality test specifically for Mersenne primes. The test states that M_p is prime if and only if

$$S_p - 2 \equiv 0 \pmod{M_p}$$

Since the relationship is if and only if, passing this test means that M_p is definitively prime. The test uses the Lucas Sequence, which is defined as

$$S_0 = 4$$

$$S_i = S_{i-1}^2 - 2, i \geq 1$$

Starting from S_0, after each iteration S_i is reduced modulo M_p. After p-2 iterations, Mp must be prime if $S_{p-2} \equiv 0 \pmod{M_p}$ holds true. This means that the Lucas-Lehmer test will take p-2 iterations (for example for $M_{136,279,841}$ the Lucas-Lehmer test alone would have taken 136,279,839 iterations) with the cost of computing each iteration itself growing iteratively. This is an expensive test but since it uses the definition of Mersenne primes, it is still significantly faster than general primality tests at this scale. Again, a fully deterministic primality test such as the Lucas-Lehmer test is needed as the Fermat Primality test is only good at eliminating a large number of composites from a batch of Mersenne prime candidates but will still produce false positives.

Since the scale of digits of the largest primes found are in the tens of millions ($M_{136,279,841}$ has 41,024,320 digits), the GNU Multiple Precision Arithmetic Library (GMP) is necessary. This repo has two C++ files. “Prime Number Generator-Limit M61.cpp” employs only the mathematical process of finding Mersenne primes as described above, but it can only find up to M_{61}. “Mersenne Prime Generator.cpp” uses GMP methods to be able to handle millions of digits.